REDES NEURAIS COM TENSORFLOW

CLASSIFICAÇÃO: REDE NEURAL PROFUNDA

DIEGO RODRIGUES DSC

INFNET

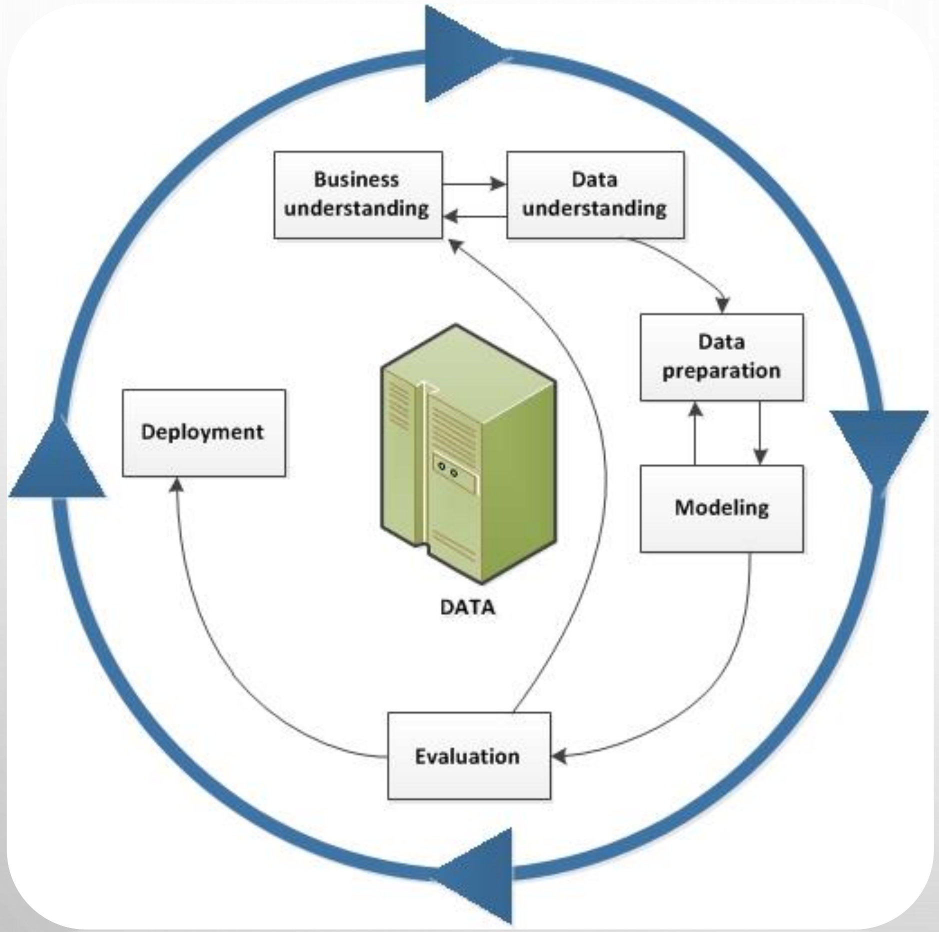
CRONOGRAMA

Dia	Aula	Trab
29/07	Perceptron de Rosenblatt	
31/07	Classificação: Neurônio Sigmóide	
05/08	Classificação: Rede Neural Feedforward	Grupos
07/08	Classificação: Rede Neural Profunda	
12/08	Regressão	Base de Dados
14/08	Agrupamento	
19/08	Séries Temporais	Modelos
21/08	Apresentação dos Trabalhos Parte I	

CLASSIFICAÇÃO : REDE NEURAL PROFUNDA

- PARTE 1 : META HEURÍSTICA DE TREINAMENTO ROBUSTA
  - BUSINESS UNDERSTANDING
    - NOVO CICLO DO CRISP
    - MODELOS MULTICLASSE
    - MÁQUINAS DE COMITÊ
  - DATA UNDERSTANDING
    - ENTROPIA E GINI
    - RELEVÂNCIA DAS VARIÁVEIS
  - MODELING
    - REDE NEURAL PROFUNDA
    - GRID SEARCH, RANDOM SEARCH & PATTERN SEARCH
    - PONTO DE OPERAÇÃO E CURVA ROC
    - TREINAMENTO ROBUSTO
  - VALIDAÇÃO
    - MATRIZ DE CONFUSÃO
- PARTE 2 : PRÁTICA
  - NOTEBOOK: CLASSIFICADOR MULTICLASSE ROBUSTO IRIS
- PARTE 3 : TRABALHOS
  - ESCOPO & EVOLUÇÃO

PARTE 1 : TEORIA



CROSS INDUSTRY PROCESS FOR DATA MINING (CRISP-DM)

BUSINESS UNDERSTANDING

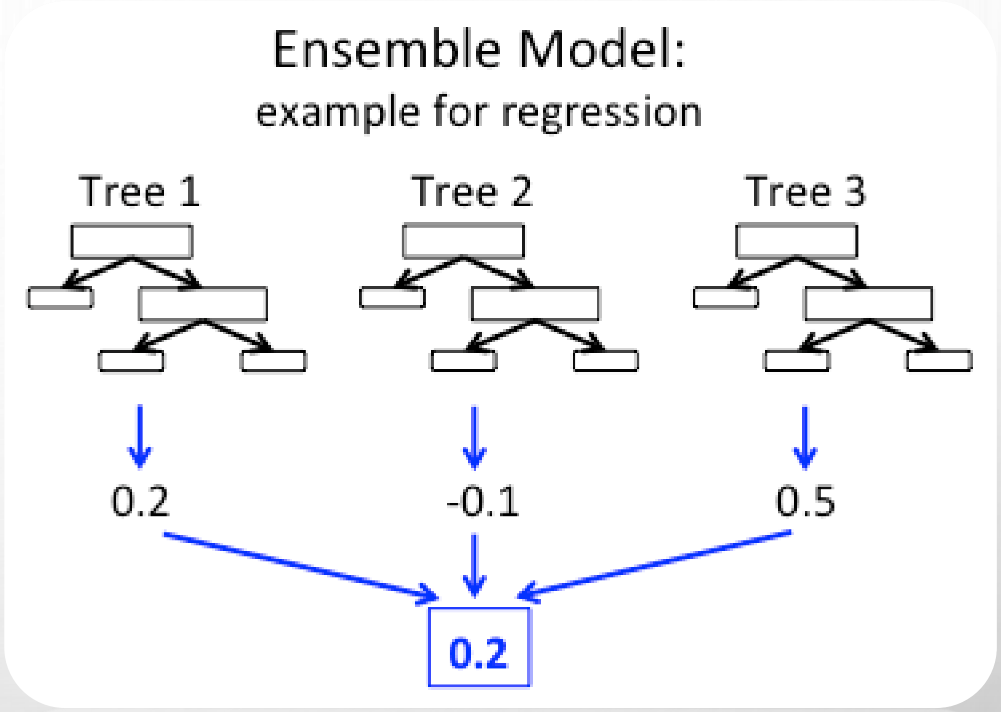
NOVO CICLO CRISP

Algoritmo	Representação	Preparação	Modelagem	Validação
Reta 2 Pontos	2D	Nenhuma	Reta 2 Pontos	Nenhuma
NN 10% VAL	2D	Nenhuma	NN Básica	Precisão/Recall
NN 5 Folds	2D	Scale	NN Hidden	Precisão/Recall
Pattern Search 5 Folds	4D / 3 Classes	Scale	NN Hidden	Acurácia

- Modelo Multiclasse
- Busca nos hiperparâmetros ótimos (# funções de ativação)
- Identificar as variáveis mais relevantes

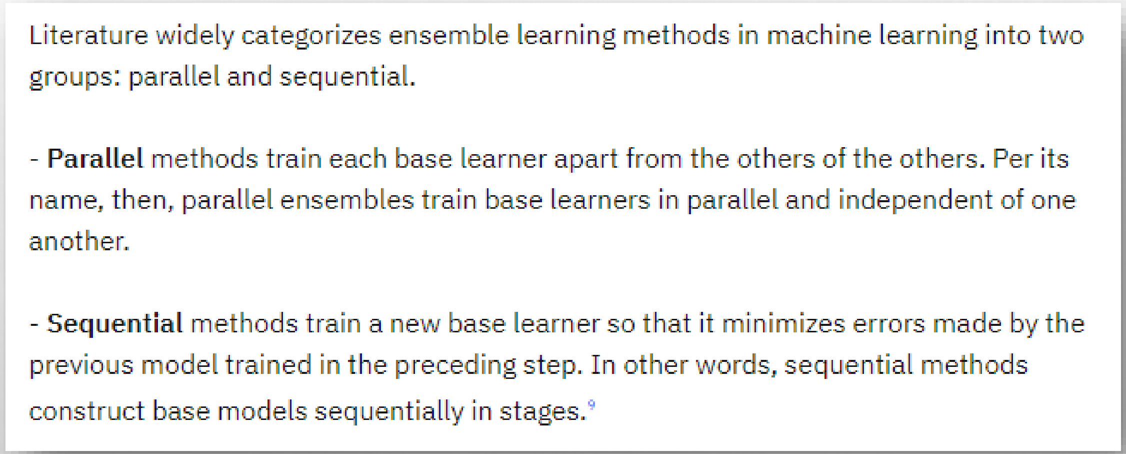
MÁQUINA DE COMITÊ (ENSEMBLE)

Algoritmos que **combinam modelos simples**, usualmente através de **votação ou ponderação**, para atingir maiores taxas de classificação.



Boa **capacidade de generalização** gerado através de **arranjos complexos** de múltiplos modelos simples de machine learning.

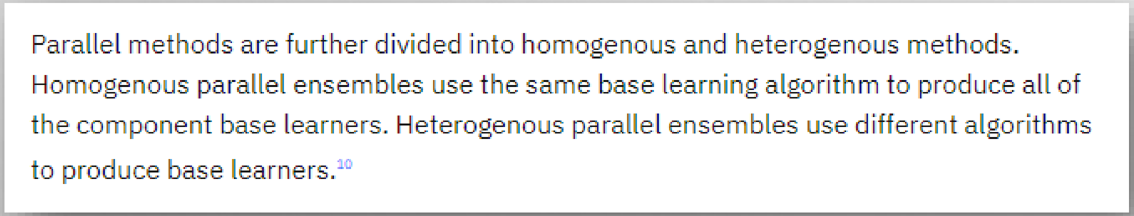
TIPOS DE COMITÊ



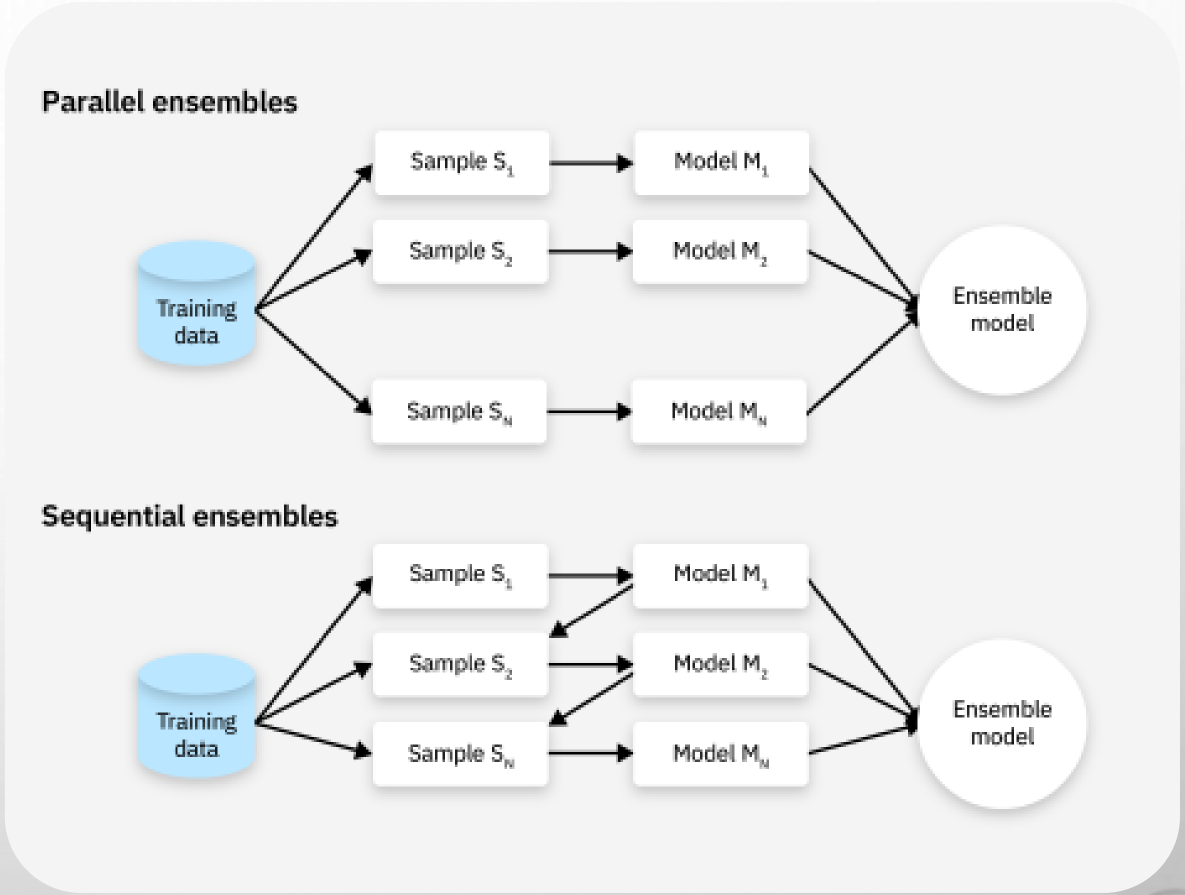
Literature widely categorizes ensemble learning methods in machine learning into two groups: parallel and sequential.

- **Parallel** methods train each base learner apart from the others of the others. Per its name, then, parallel ensembles train base learners in parallel and independent of one another.

- **Sequential** methods train a new base learner so that it minimizes errors made by the previous model trained in the preceding step. In other words, sequential methods construct base models sequentially in stages.[9]

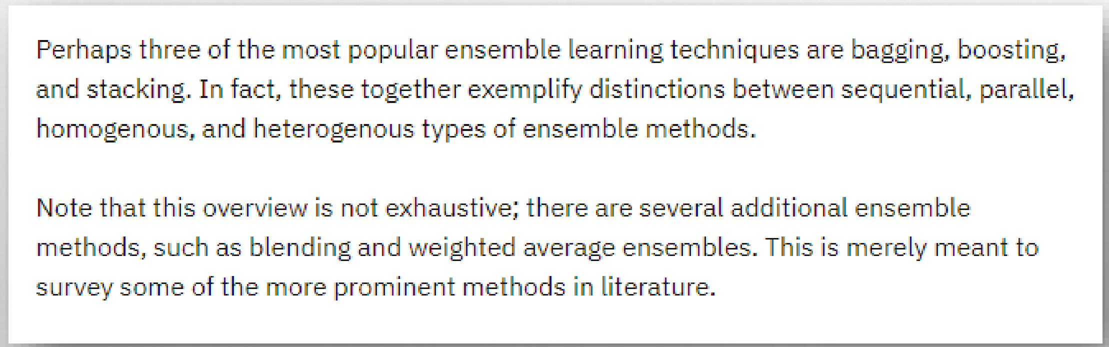


Parallel methods are further divided into homogenous and heterogenous methods. Homogenous parallel ensembles use the same base learning algorithm to produce all of the component base learners. Heterogenous parallel ensembles use different algorithms to produce base learners.[10]



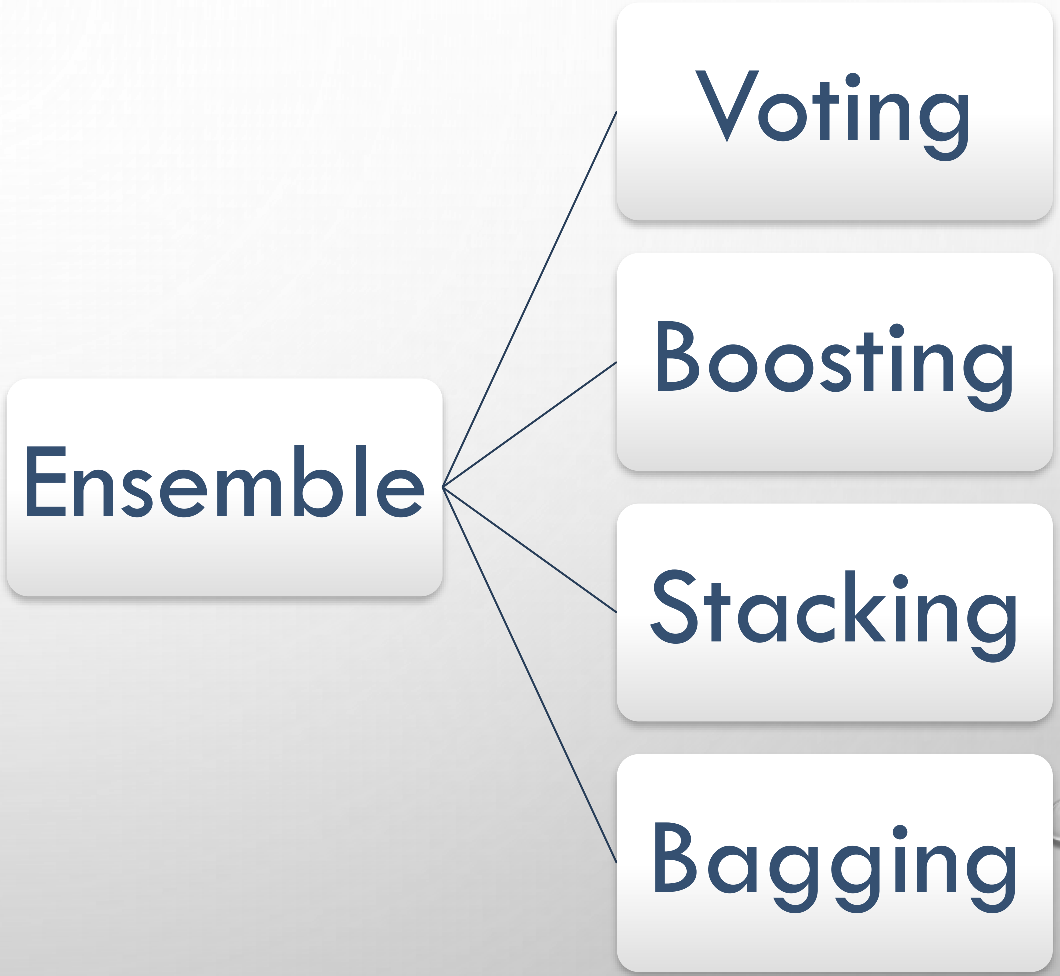
https://www.ibm.com/topics/ensemble-learning

TÉCNICAS DE TREINAMENTO



Perhaps three of the most popular ensemble learning techniques are bagging, boosting, and stacking. In fact, these together exemplify distinctions between sequential, parallel, homogenous, and heterogenous types of ensemble methods.

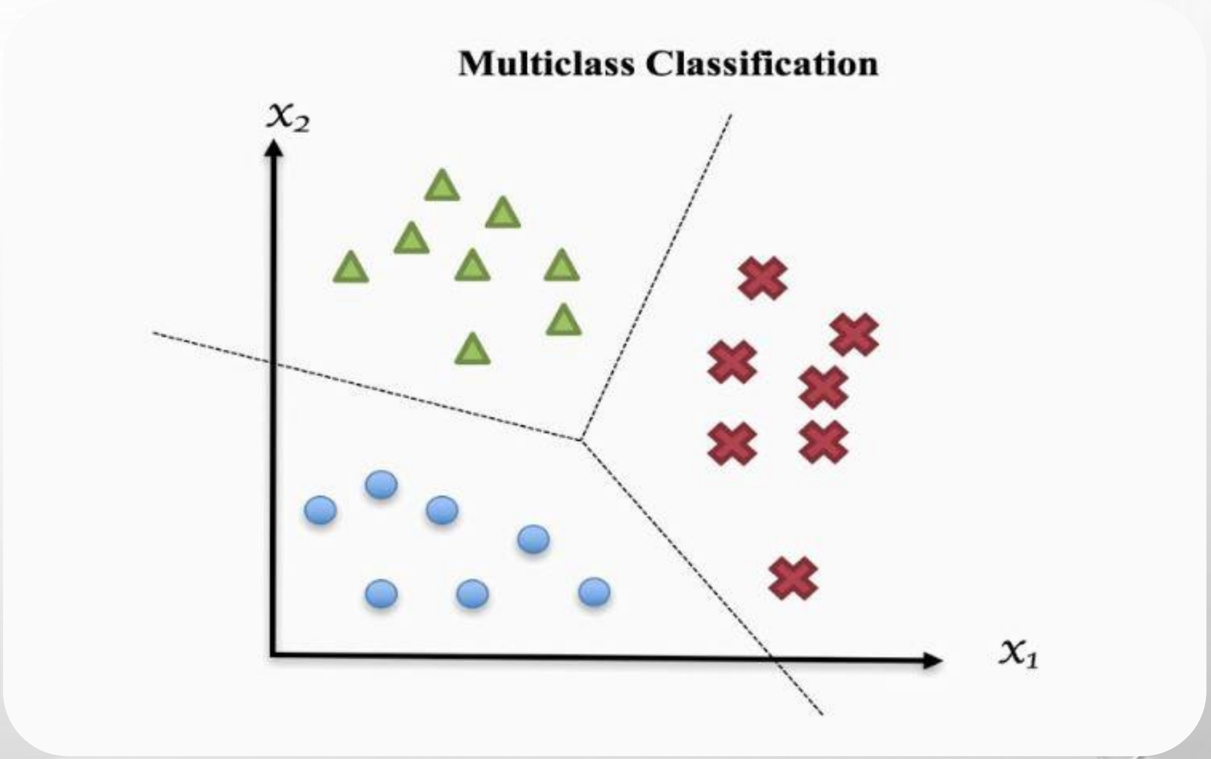
Note that this overview is not exhaustive; there are several additional ensemble methods, such as blending and weighted average ensembles. This is merely meant to survey some of the more prominent methods in literature.



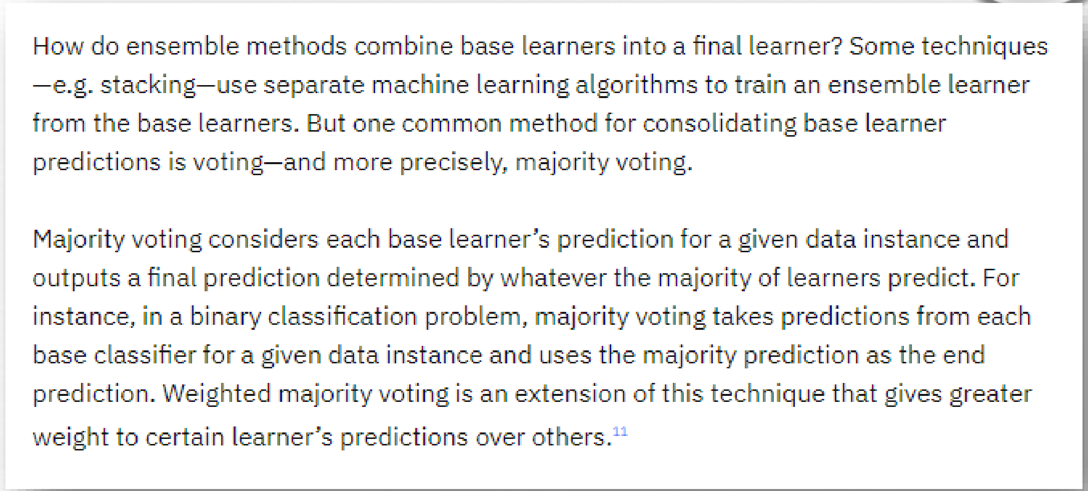
MÁQUINA DE COMITÊ - VOTAÇÃO

Modelos de Votação

- Discriminar múltiplos objetos em paralelo.
- Quaisquer modelos podem ser combinados.
- Extensão para o caso multiclasse de maneira simples e intuitiva.

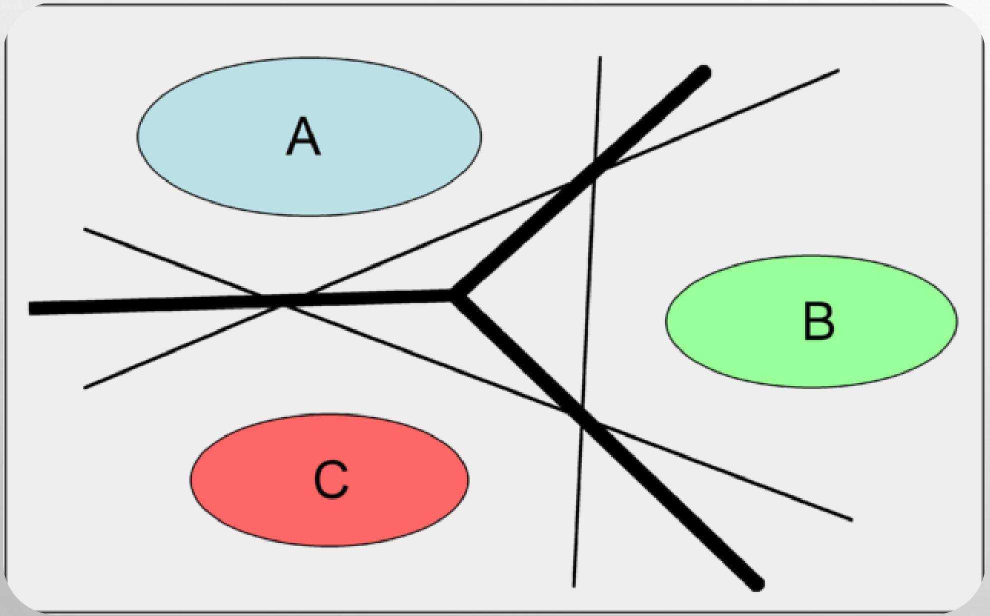


VOTAÇÃO

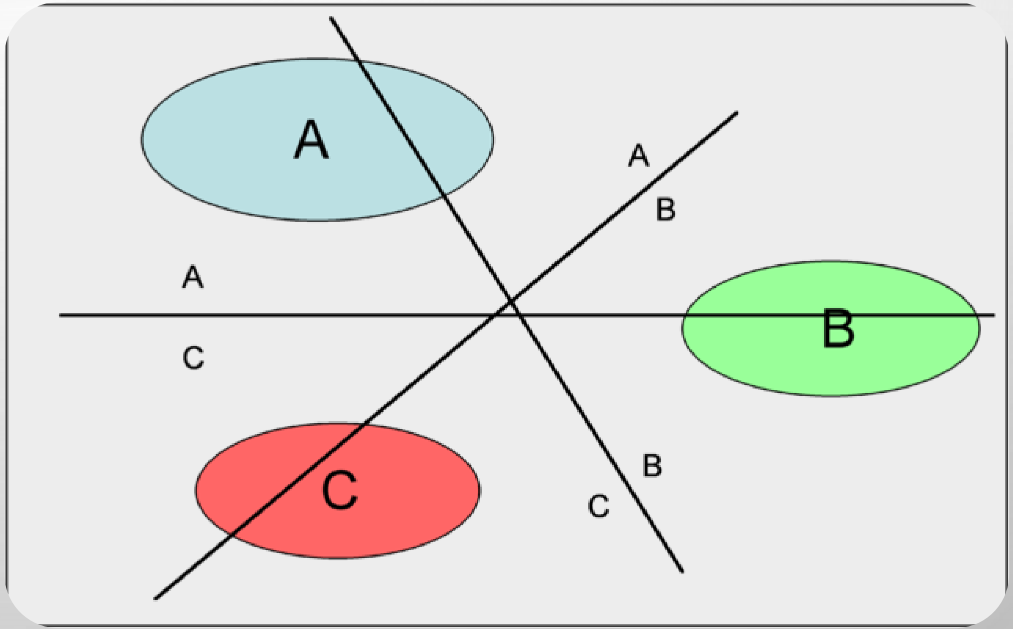


How do ensemble methods combine base learners into a final learner? Some techniques—e.g. stacking—use separate machine learning algorithms to train an ensemble learner from the base learners. But one common method for consolidating base learner predictions is voting—and more precisely, majority voting.

Majority voting considers each base learner's prediction for a given data instance and outputs a final prediction determined by whatever the majority of learners predict. For instance, in a binary classification problem, majority voting takes predictions from each base classifier for a given data instance and uses the majority prediction as the end prediction. Weighted majority voting is an extension of this technique that gives greater weight to certain learner's predictions over others.[11]

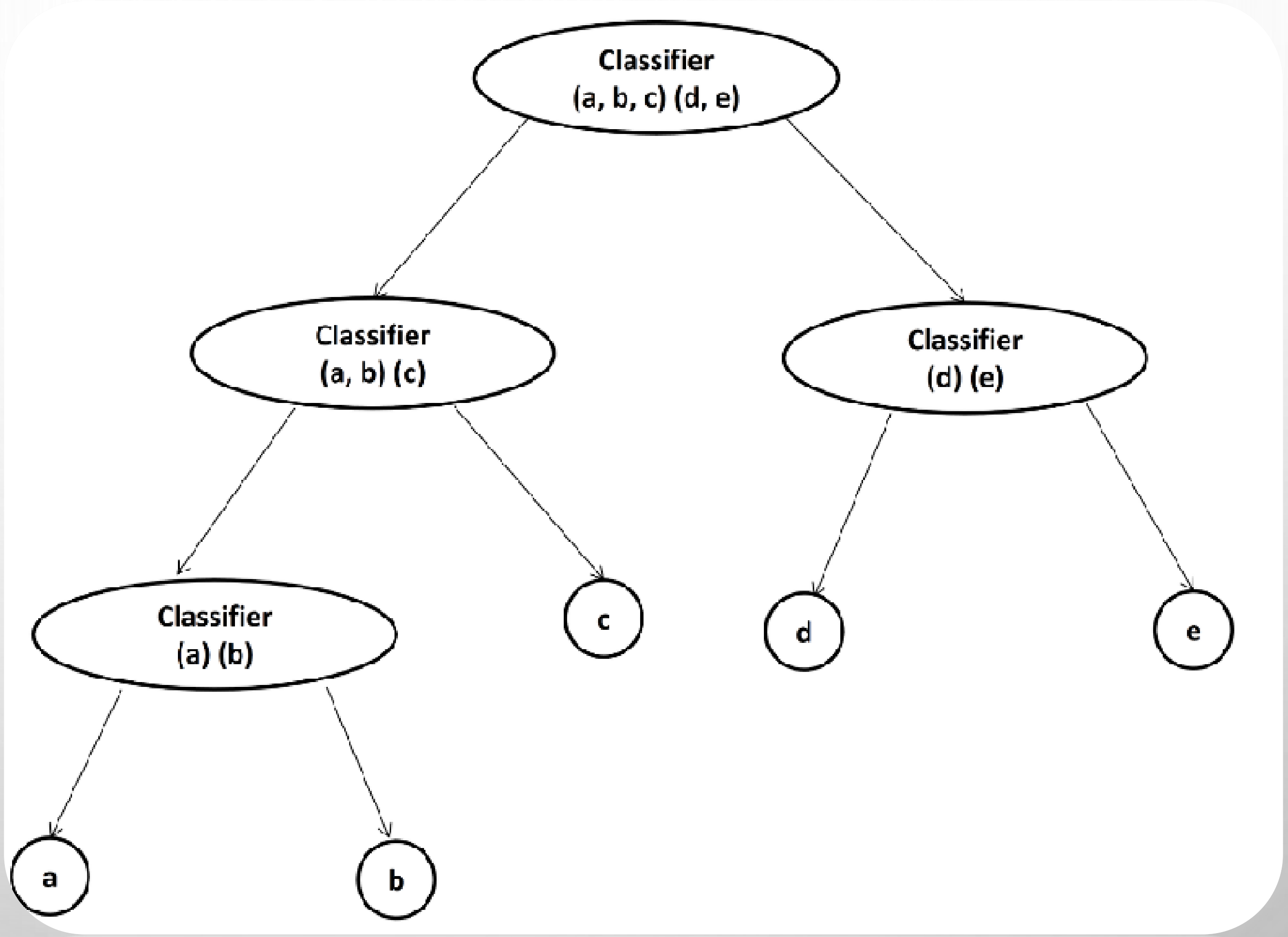


ONE AGAINST ALL

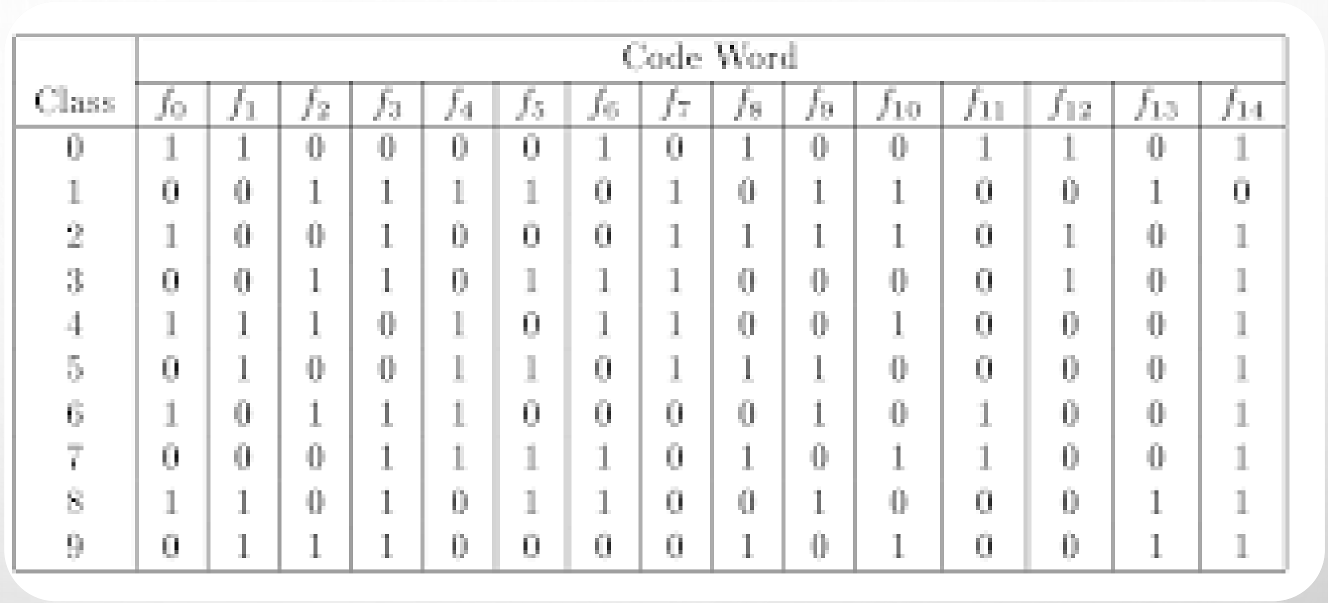


ONE AGAINST ONE

VOTAÇÃO: DIRECTED ACYCLIC GRAPH (DAG)



ERROR CORRECTING CODES



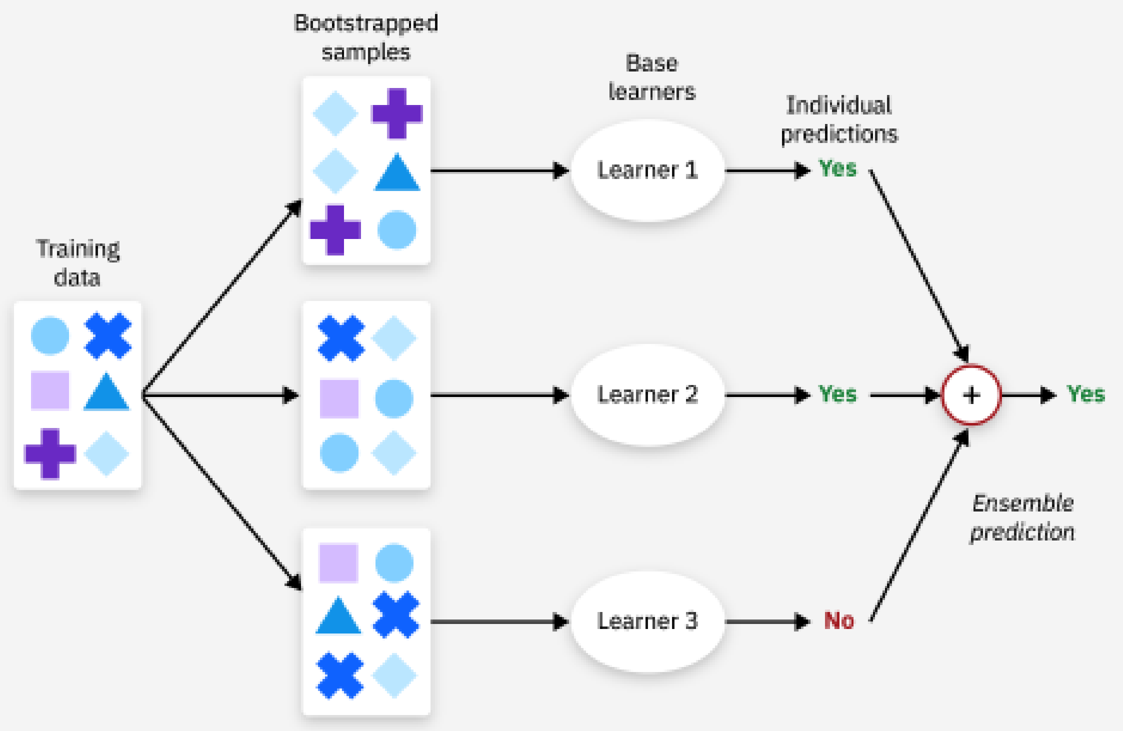
Class	Code Word														
	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}
0	1	1	0	0	0	0	1	0	1	0	0	1	1	0	1
1	0	0	1	1	1	1	0	1	0	1	1	0	0	1	0
2	1	0	0	1	0	0	0	1	1	1	1	0	1	0	1
3	0	0	1	1	0	1	1	1	0	0	0	0	1	0	1
4	1	1	1	0	1	0	1	1	0	0	1	0	0	0	1
5	0	1	0	0	1	1	0	1	1	1	0	0	0	0	1
6	1	0	1	1	1	0	0	0	0	1	0	1	0	0	1
7	0	0	0	1	1	1	1	0	1	0	1	1	0	0	1
8	1	1	0	1	0	1	1	0	0	1	0	0	0	1	1
9	0	1	1	1	0	0	0	0	1	0	1	0	0	1	1

BAGGING

Bagging is a homogenous parallel method sometimes called *bootstrap aggregating*. It uses modified replicates of a given training data set to train multiple base learners with the same training algorithm.[12] Scikit-learn's ensemble module in Python contains functions for implementing bagging, such as BaggingClassifier.

More specifically, bagging uses a technique called bootstrap resampling to derive multiple new datasets from one initial training dataset in order to train multiple base learners. How does this work? Say a training dataset contains *n* training examples. Bootstrap resampling copies *n* data instances from that set into a new subsample dataset, with some initial instances appearing more than once and others excluded entirely. These are bootstrap samples. Repeating this process *x* times produces *x* iterations of the original dataset, each containing *n* samples from the initial set. Each iteration of the initial set is then used to train a separate base learner with the same learning algorithm.[13]

Random forest is an extension of bagging that specifically denotes the use of bagging to construct ensembles of randomized decision trees. This differs from standard decision trees in that the latter samples every feature to identify the best for splitting. By contrast, random forests iteratively sample random subsets of features to create a decision node.[14]

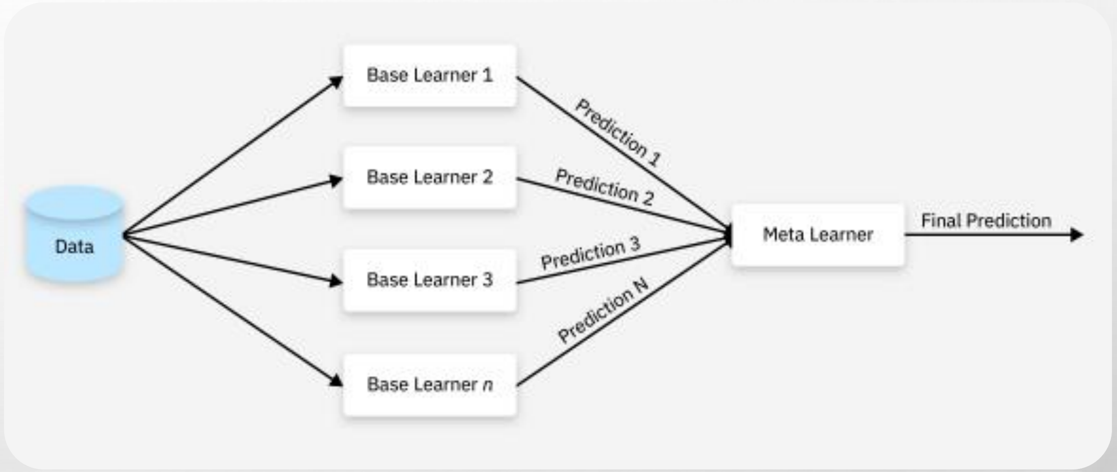


STACKING

Stacking, or stacked generalization,[15] is a heterogenous parallel method that exemplifies what is known as meta-learning. Meta-learning consists of training a meta-learner from the output of multiple base learners. Stacking specifically trains several base learners from the same dataset using a different training algorithm for each learner. Each base learner makes predictions on an unseen dataset. These first model predictions are then compiled and used to train a final model, being the meta-model.[16]

Note the importance of using a different dataset from that used to train the base learners in order to train the meta-learner. Using the same dataset to train the base learners and the meta-learner can result in overfitting. This can require excluding data instances from the base learner training data to serve as its test set data, which in turn becomes training data for the meta-learner. Literature often recommends techniques such as cross-validation to ensure these datasets do not overlap.[17]

Much as bagging, the sklearn.ensemble module in Python provides various functions for implementing stacking techniques.



BOOSTING

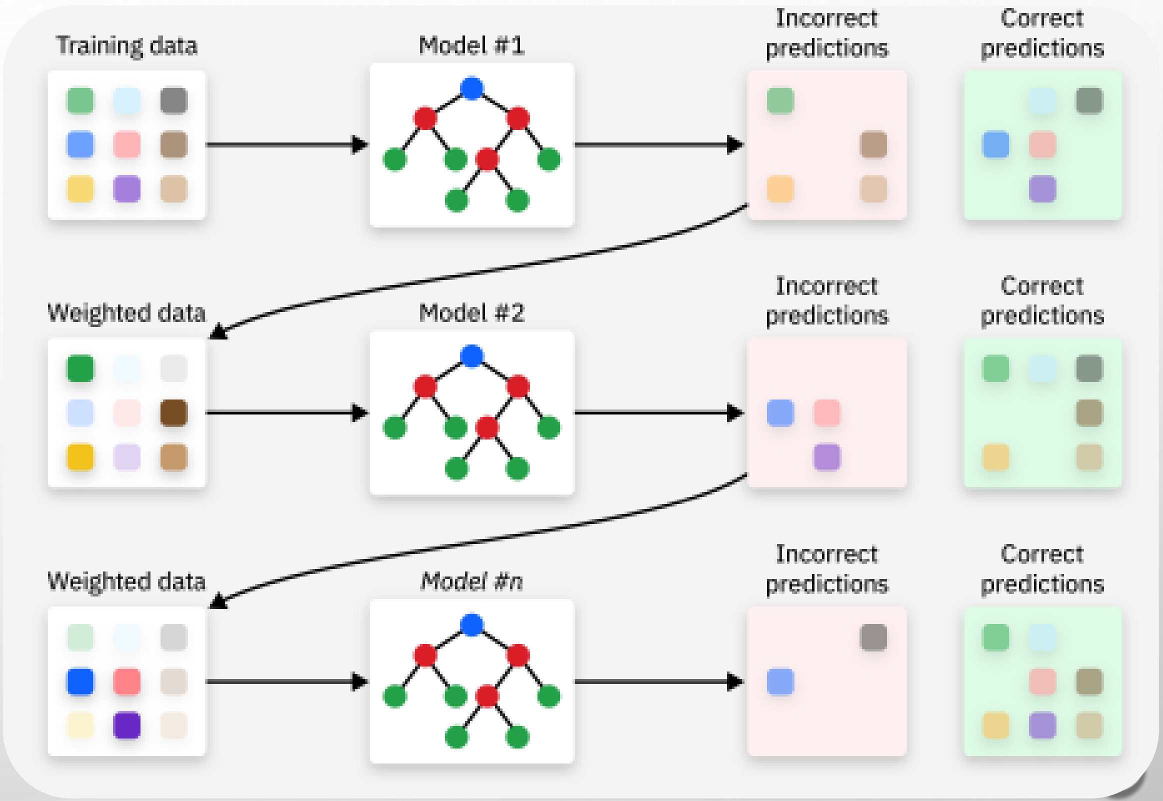
Boosting algorithms are a sequential ensemble method. Boosting has many variations, but they all follow the same general procedure. Boosting trains a learner on some initial dataset, *d*. The resultant learner is typically weak, misclassifying many samples in the dataset. Much like bagging, boosting then samples instances from the initial dataset to create a new dataset (d_2). Unlike bagging, however, boosting prioritizes misclassified data instances from the first model or learner. A new learner is trained on this new dataset d_2. Then a third dataset (d_3) is then compiled from d_1 and d_2, prioritizes the second learner's misclassified samples and instances in which d_1 and d_2 disagree. The process repeats *n* times to produce *n* learners. Boosting then combines and weights the all the learners together to produce final predictions.[18]

Boosting algorithms largely differ in how they prioritize erroneously predicted data instances when creating a new dataset. Two of the most prominent boosting methods may illustrate this:

- **Adaptive boosting** (AdaBoost) weights model errors. That is, when creating a new iteration of a dataset for training the next learner, AdaBoost adds weights to the previous learner's misclassified samples, causing the next learner to prioritize those misclassified samples.

- **Gradient boosting** uses residual errors when training new learners. Rather than weight misclassified samples, gradient boosting uses residual errors from a previous model to set target predictions for the next model. In this way, it attempts to close the gap of error left by one model.[19]

Unfortunately, sklearn contains no pre-defined functions for implementing boosting. The Extreme Gradient Boosting (XGBoost) open-source library, however, provides code for implementing gradient boosting in Python.



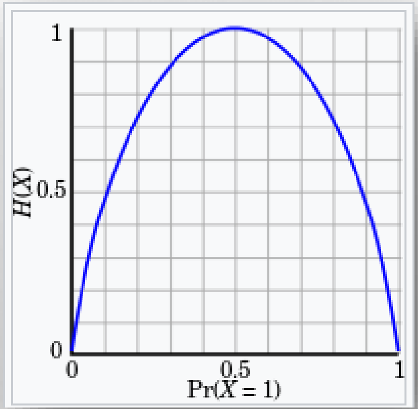
DATA UNDERSTANDING & PREPARATION

TÉCNICAS DE SELEÇÃO DE ATRIBUTOS

Wrapper – mede a relação entre atributos e classes, utilizando um modelo treinado.

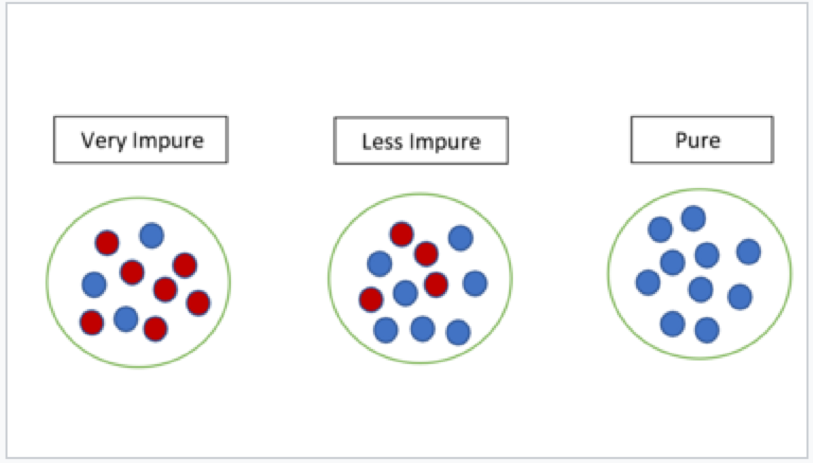
- **Gini –** Estatística que representa a importância de um atributo na divisão da base de dados por uma árvore de decisão.
- **Relevância** – Estatística que representa a variação causada na saída do modelo quando um atributo é substituído por sua média.

ENTROPIA E GANHO DE INFORMAÇÃO

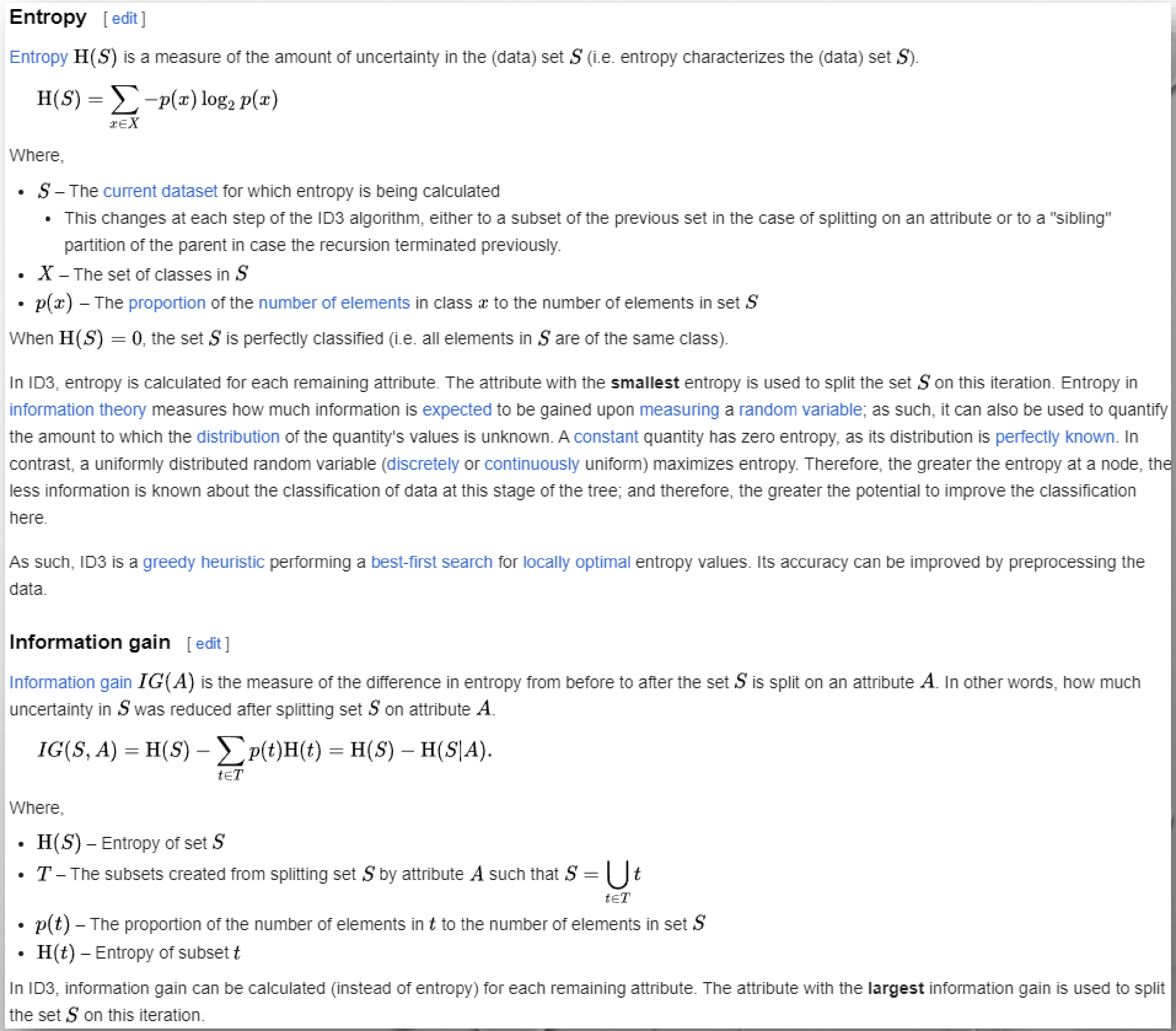


Entropy $H(X)$ (i.e. the expected surprisal) of a coin flip, measured in bits, graphed versus the bias of the coin $\Pr(X = 1)$, where $X = 1$ represents a result of heads.[10]: 14–15

Here, the entropy is at most 1 bit, and to communicate the outcome of a coin flip (2 possible values) will require an average of at most 1 bit (exactly 1 bit for a fair coin). The result of a fair die (6 possible values) would have entropy $\log_2 6$ bits.



Entropy diagram[2]



Entropy [edit]

Entropy $\mathrm{H}(S)$ is a measure of the amount of uncertainty in the (data) set S (i.e. entropy characterizes the (data) set S).

$$\mathrm{H}(S) = \sum_{x \in X} -p(x) \log_2 p(x)$$

Where,

- S – The current dataset for which entropy is being calculated
  - This changes at each step of the ID3 algorithm, either to a subset of the previous set in the case of splitting on an attribute or to a "sibling" partition of the parent in case the recursion terminated previously.
- X – The set of classes in S
- $p(x)$ – The proportion of the number of elements in class x to the number of elements in set S

When $\mathrm{H}(S) = 0$, the set S is perfectly classified (i.e. all elements in S are of the same class).

In ID3, entropy is calculated for each remaining attribute. The attribute with the **smallest** entropy is used to split the set S on this iteration. Entropy in information theory measures how much information is expected to be gained upon measuring a random variable; as such, it can also be used to quantify the amount to which the distribution of the quantity's values is unknown. A constant quantity has zero entropy, as its distribution is perfectly known. In contrast, a uniformly distributed random variable (discretely or continuously uniform) maximizes entropy. Therefore, the greater the entropy at a node, the less information is known about the classification of data at this stage of the tree; and therefore, the greater the potential to improve the classification here.

As such, ID3 is a greedy heuristic performing a best-first search for locally optimal entropy values. Its accuracy can be improved by preprocessing the data.

Information gain [edit]

Information gain $IG(A)$ is the measure of the difference in entropy from before to after the set S is split on an attribute A. In other words, how much uncertainty in S was reduced after splitting set S on attribute A.

$$IG(S, A) = \mathrm{H}(S) - \sum_{t \in T} p(t)\mathrm{H}(t) = \mathrm{H}(S) - \mathrm{H}(S|A).$$

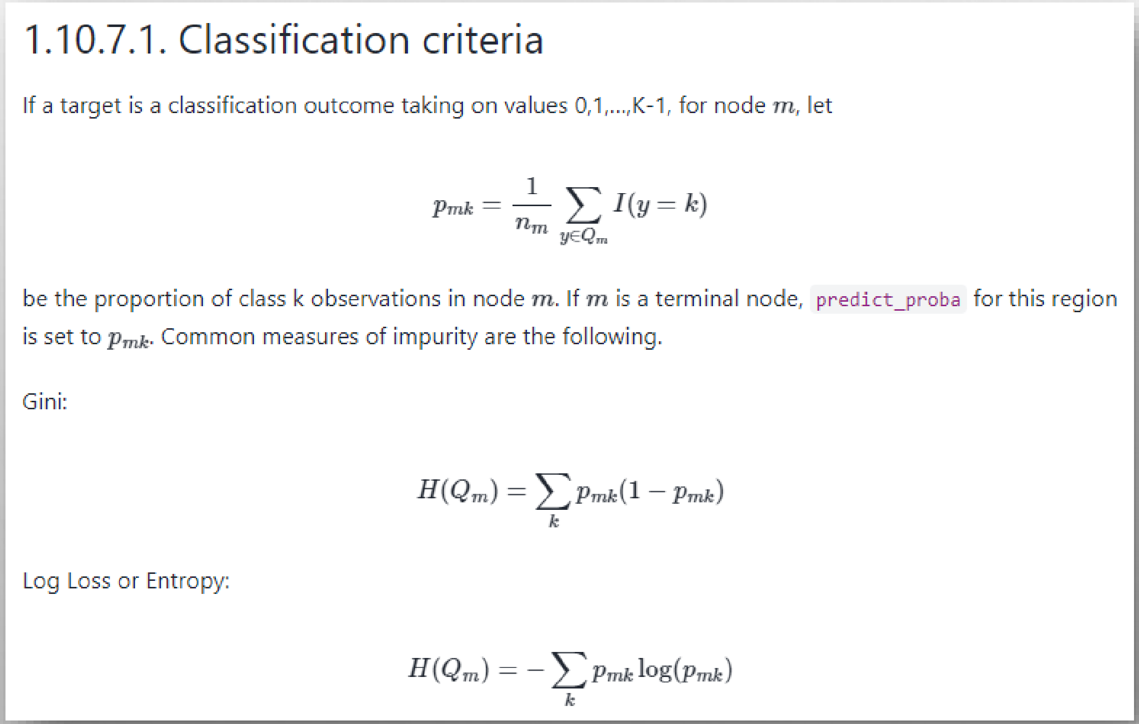
Where,

- $\mathrm{H}(S)$ – Entropy of set S
- T – The subsets created from splitting set S by attribute A such that $S = \bigcup_{t \in T} t$
- $p(t)$ – The proportion of the number of elements in t to the number of elements in set S
- $\mathrm{H}(t)$ – Entropy of subset t

In ID3, information gain can be calculated (instead of entropy) for each remaining attribute. The attribute with the **largest** information gain is used to split the set S on this iteration.

IMPUREZA DE GINI

$$Gini = 1 - \sum_j p_j^2$$



1.10.7.1. Classification criteria

If a target is a classification outcome taking on values 0,1,...,K-1, for node m, let

$$p_{mk} = \frac{1}{n_m} \sum_{y \in Q_m} I(y = k)$$

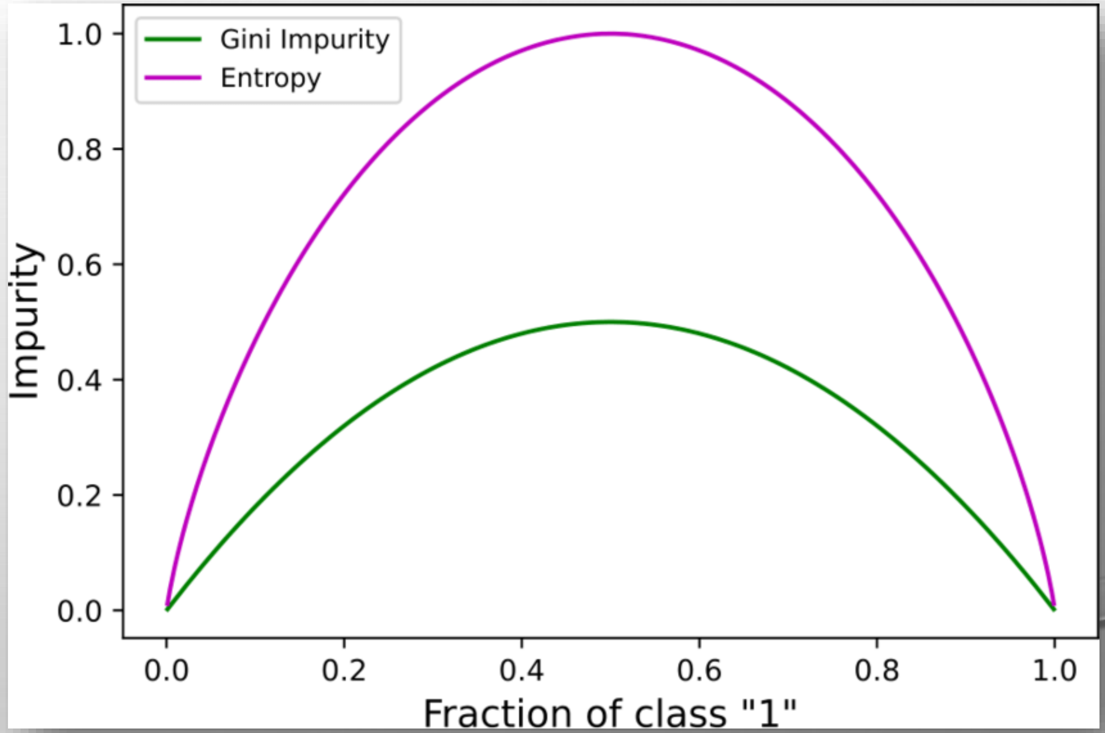
be the proportion of class k observations in node m. If m is a terminal node, `predict_proba` for this region is set to p_{mk}. Common measures of impurity are the following.

Gini:

$$H(Q_m) = \sum_k p_{mk}(1 - p_{mk})$$

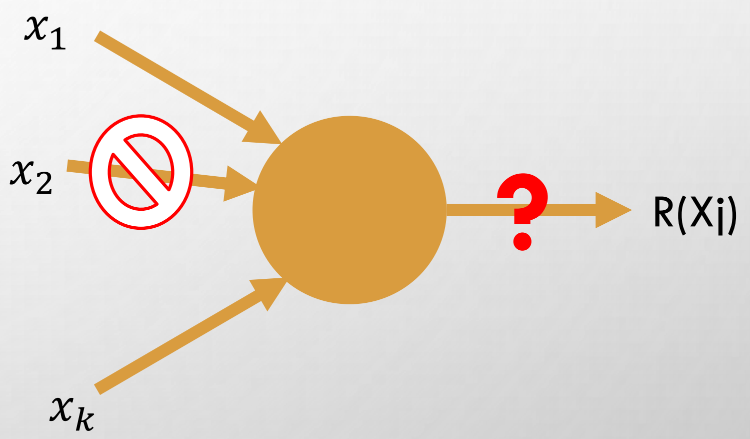
Log Loss or Entropy:

$$H(Q_m) = - \sum_k p_{mk} \log(p_{mk})$$



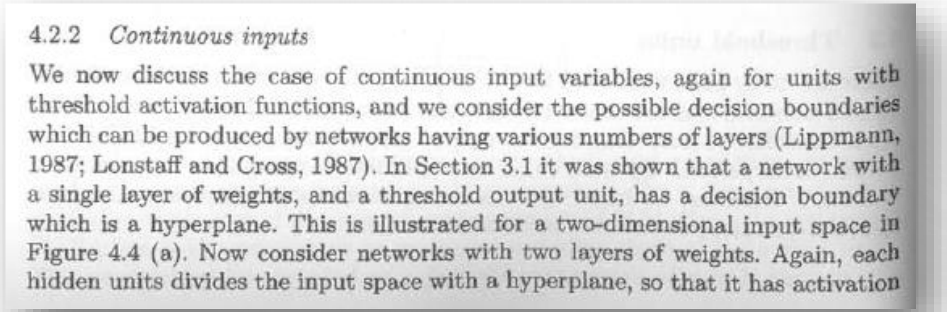
RELEVÂNCIA

$$R(X_j) = \frac{\sum_{i=1}^{N} \left|\left|\hat{y}(\mathbf{x_i}) - \hat{y}(\mathbf{x_i}|_{x_{ij}=\bar{x}_j})\right|\right|^2}{N}$$



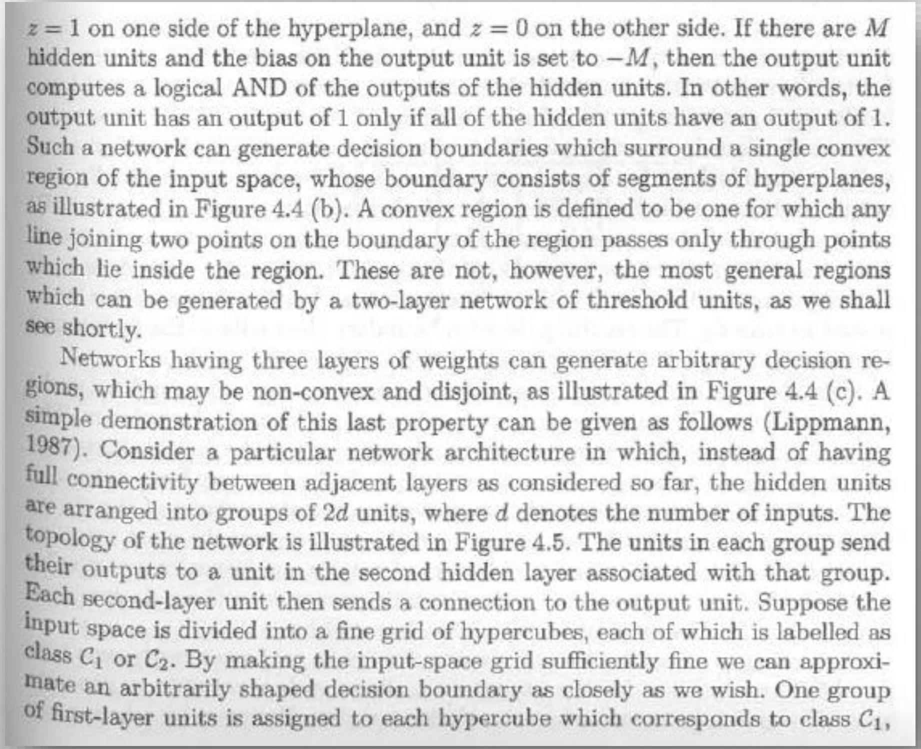
MODELING

REDE NEURAL PROFUNDA



4.2.2 *Continuous inputs*

We now discuss the case of continuous input variables, again for units with threshold activation functions, and we consider the possible decision boundaries which can be produced by networks having various numbers of layers (Lippmann, 1987; Lonstaff and Cross, 1987). In Section 3.1 it was shown that a network with a single layer of weights, and a threshold output unit, has a decision boundary which is a hyperplane. This is illustrated for a two-dimensional input space in Figure 4.4 (a). Now consider networks with two layers of weights. Again, each hidden units divides the input space with a hyperplane, so that it has activation



$z = 1$ on one side of the hyperplane, and $z = 0$ on the other side. If there are M hidden units and the bias on the output unit is set to $-M$, then the output unit computes a logical AND of the outputs of the hidden units. In other words, the output unit has an output of 1 only if all of the hidden units have an output of 1. Such a network can generate decision boundaries which surround a single convex region of the input space, whose boundary consists of segments of hyperplanes, as illustrated in Figure 4.4 (b). A convex region is defined to be one for which any line joining two points on the boundary of the region passes only through points which lie inside the region. These are not, however, the most general regions which can be generated by a two-layer network of threshold units, as we shall see shortly.

Networks having three layers of weights can generate arbitrary decision regions, which may be non-convex and disjoint, as illustrated in Figure 4.4 (c). A simple demonstration of this last property can be given as follows (Lippmann, 1987). Consider a particular network architecture in which, instead of having full connectivity between adjacent layers as considered so far, the hidden units are arranged into groups of $2d$ units, where d denotes the number of inputs. The topology of the network is illustrated in Figure 4.5. The units in each group send their outputs to a unit in the second hidden layer associated with that group. Each second-layer unit then sends a connection to the output unit. Suppose the input space is divided into a fine grid of hypercubes, each of which is labelled as class $\mathcal{C}_1$ or $\mathcal{C}_2$. By making the input-space grid sufficiently fine we can approximate an arbitrarily shaped decision boundary as closely as we wish. One group of first-layer units is assigned to each hypercube which corresponds to class $\mathcal{C}_1$,

Neural Networks for Pattern Recognition, Bishop

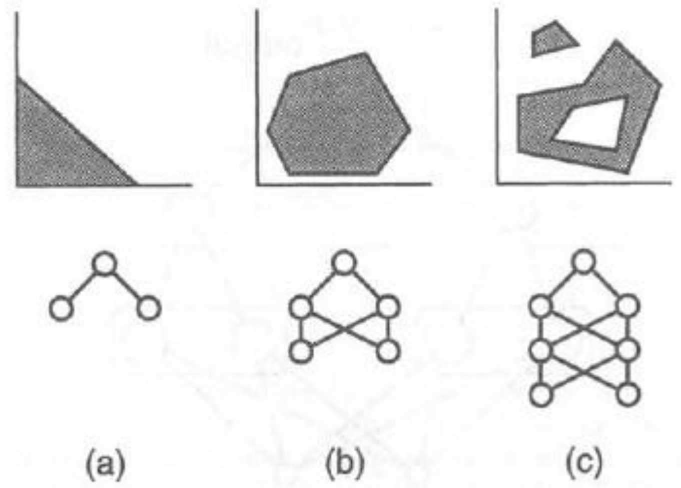
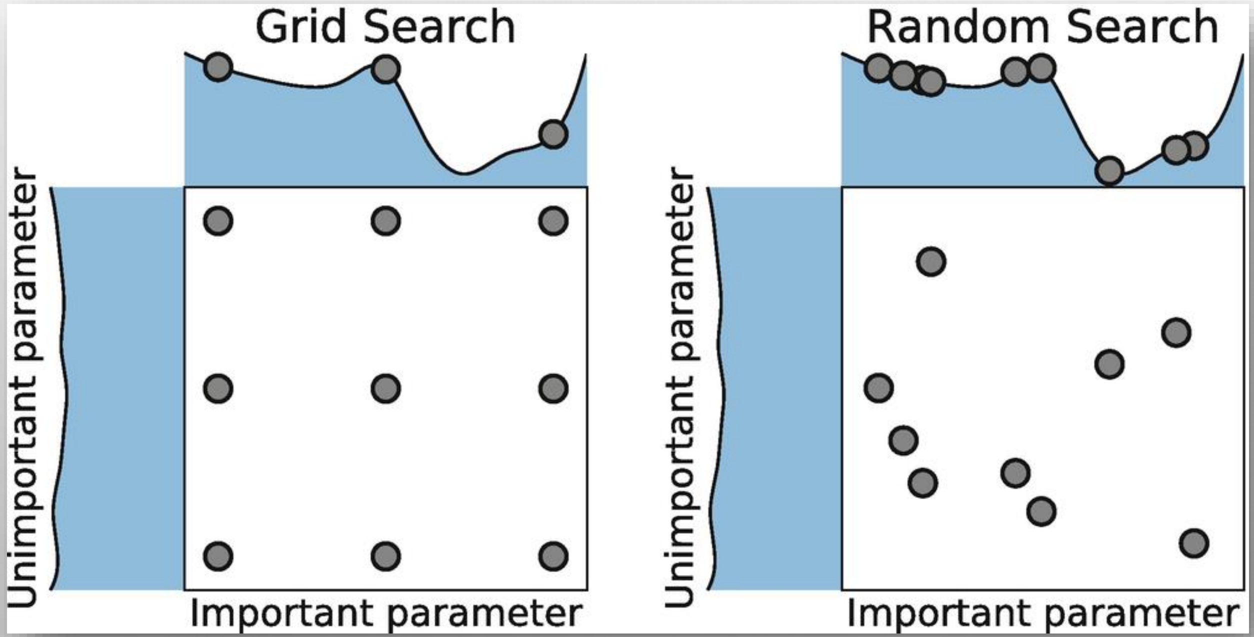
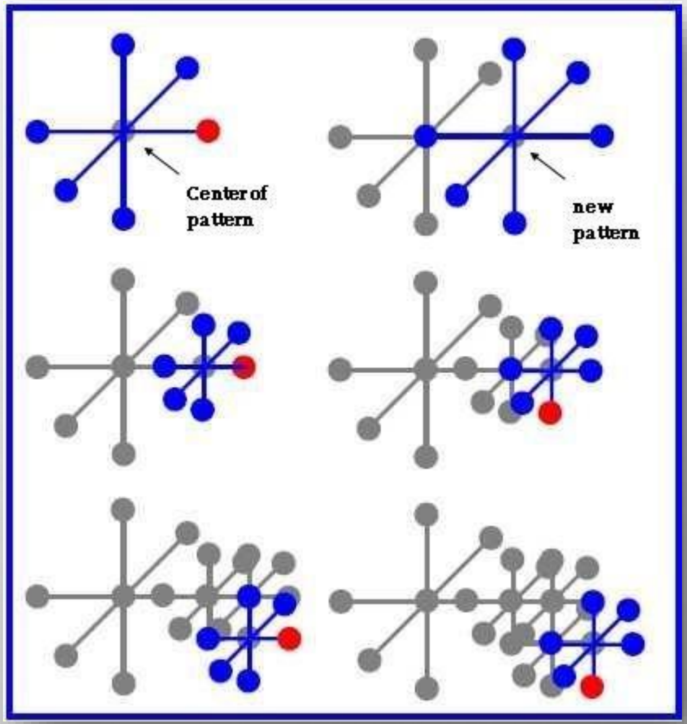


Figure 4.4. Illustration of some possible decision boundaries which can be generated by networks having threshold activation functions and various numbers of layers. Note that, for the two-layer network in (b), a single convex region of the form shown is not the most general possible.

OTIMIZAÇÃO DE HIPERPARÂMETROS



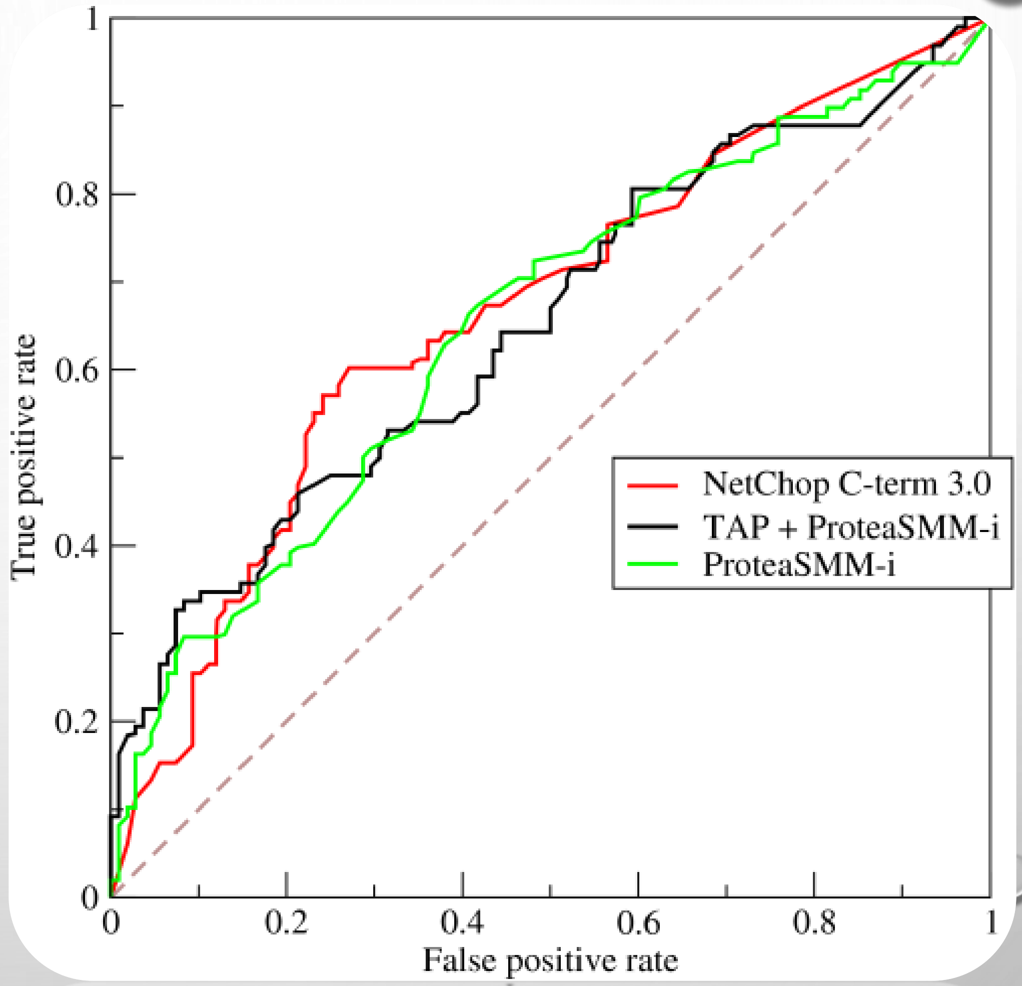
GRID SEARCH



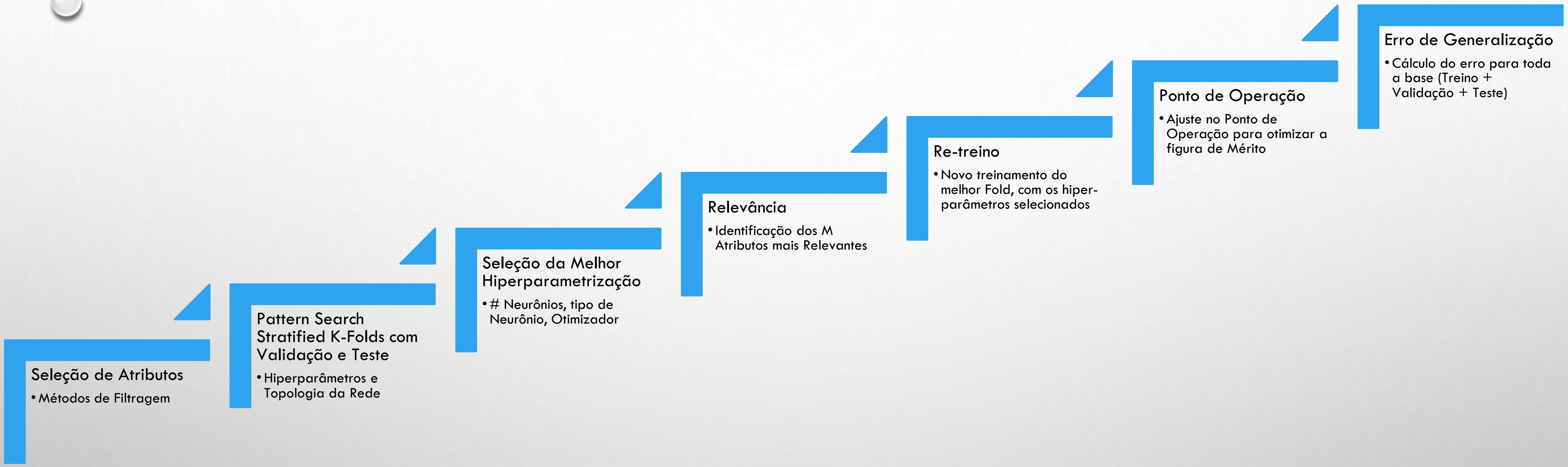
PATTERN SEARCH

PONTO DE OPERAÇÃO

- **Curva ROC**
  - Calibra a saída do modelo, ajudando a configurar o ponto de operação entre Precisão / Recall / Acurácia.



TREINAMENTO ROBUSTO



Seleção de Atributos

- Métodos de Filtragem

Pattern Search Stratified K-Folds com Validação e Teste

- Hiperparâmetros e Topologia da Rede

Seleção da Melhor Hiperparametrização

- # Neurônios, tipo de Neurônio, Otimizador

Relevância

- Identificação dos M Atributos mais Relevantes

Re-treino

- Novo treinamento do melhor Fold, com os hiper-parâmetros selecionados

Ponto de Operação

- Ajuste no Ponto de Operação para otimizar a figura de Mérito

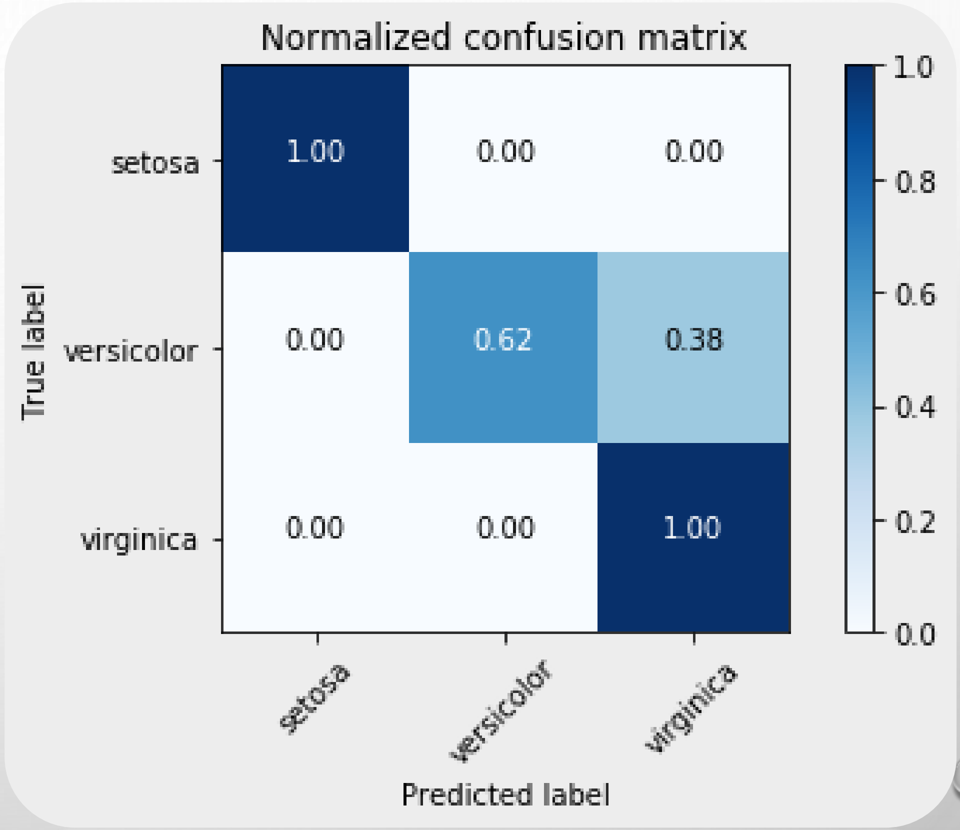
Erro de Generalização

- Cálculo do erro para toda a base (Treino + Validação + Teste)

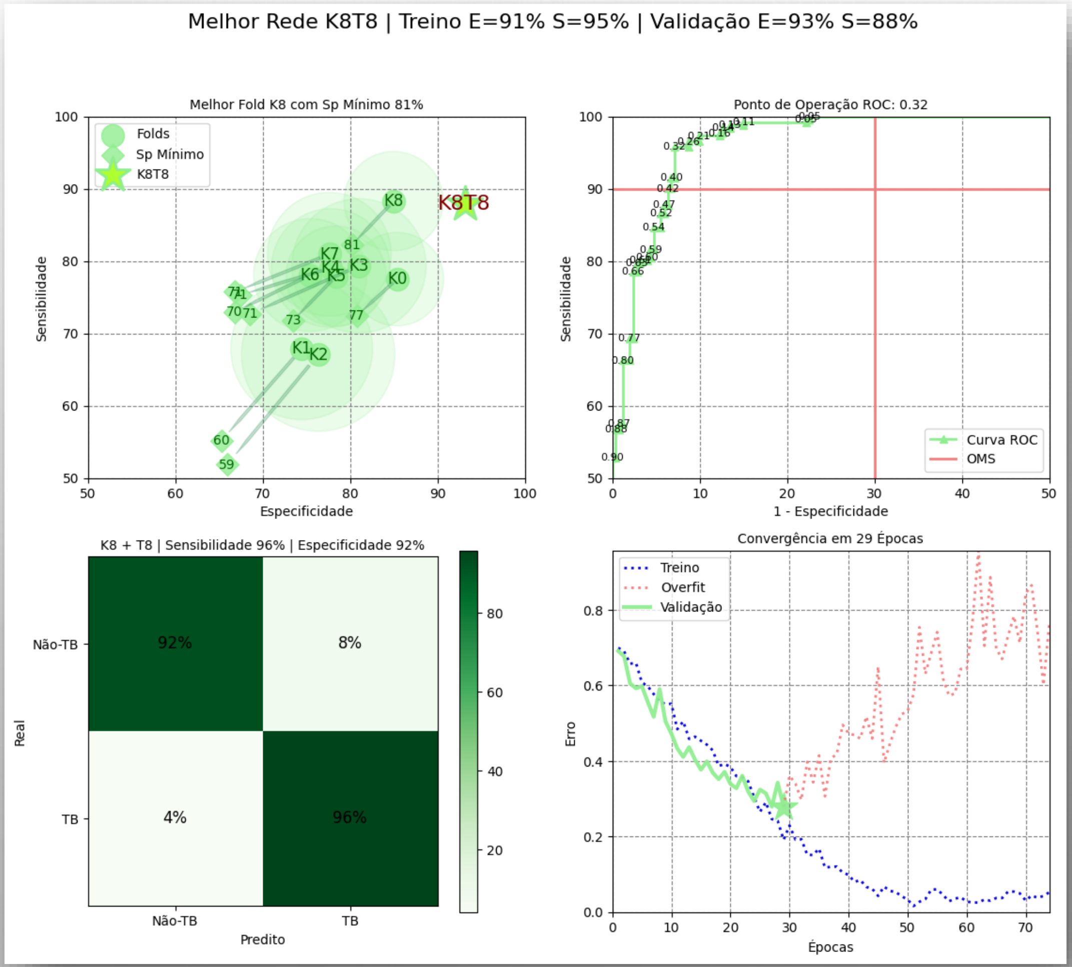
VALIDATION

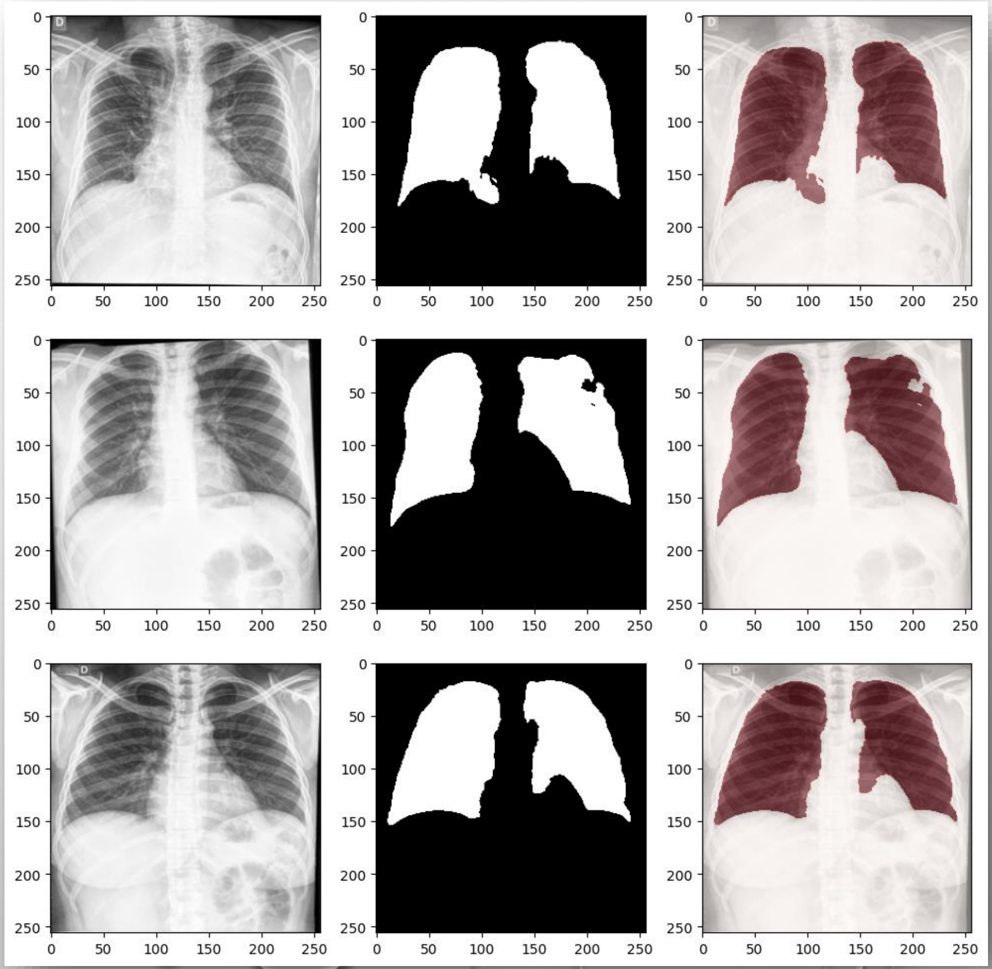
MATRIZ DE CONFUSÃO

Comparação entre o resultado do classificador para as diferentes classes.



VALIDAÇÃO DO TREINAMENTO : SEGMENTADOR DE PULMÕES





PARTE 2 : PRÁTICA

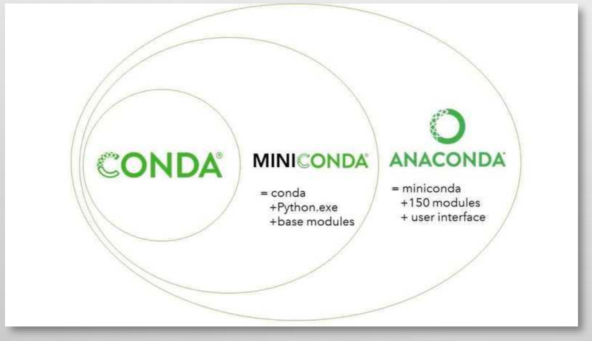
AMBIENTE PYTHON

4. Variáveis Aleatórias

5. Visualização

6. Machine Learning

1. Editor de Código



2. Gestor de Ambiente

3. Ambiente Python do Projeto

3. Notebook Dinâmico

PROBLEMA DE NEGÓCIO

Características das flores

Largura & comprimento da pétala

Largura & comprimento da sépala

Iris Setosa

Iris Versicolor

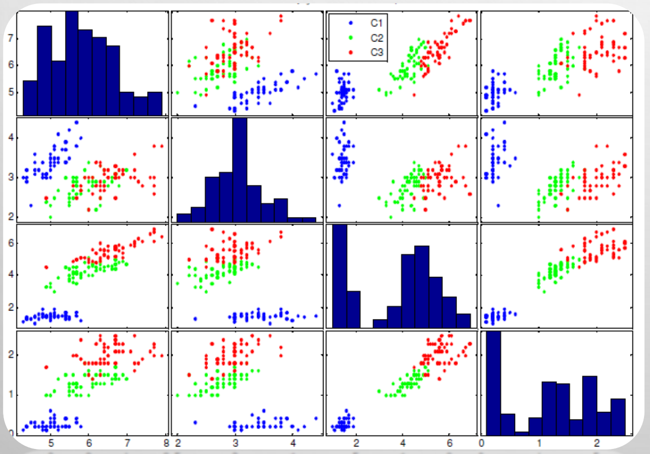
Iris Virginica

REPRESENTAÇÃO

Características das flores

Largura & comprimento da pétala

Largura & comprimento da sépala

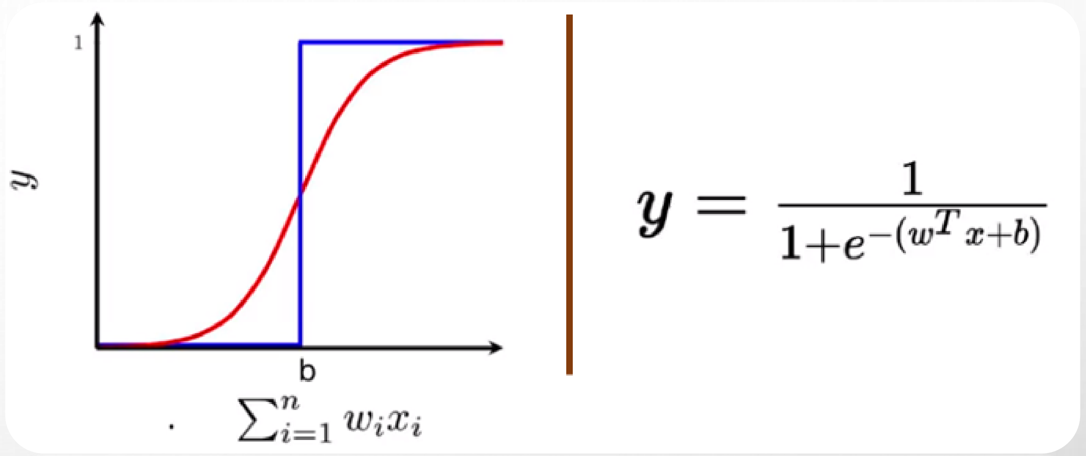


http://archive.ics.uci.edu/ml/datasets/Iris

Espaço de atributos com **4 dimensões!**

MODELAGEM

- REDE NEURAL FEED FORWARD
  - REPRESENTAÇÃO: 4 ATRIBUTOS > 2 ATRIBUTOS MAIS RELEVANTES
  - HIPERPARÂMETROS: PATTERN SEARCH NO # DE NEURÔNIOS DE CADA TIPO NA CAMADA OCULTA.
  - TREINAMENTO: BASE DE TREINO COMPLETA.
    - ACURÁCIA
    - VALIDAÇÃO CRUZADA 10 FOLDS



$$y = \frac{1}{1+e^{-(w^T x+b)}}$$

CLASSIFICADOR IRIS

EXERCÍCIO: TREINAMENTO PATTERN SEARCH

PRÓXIMA AULA: REGRESSÃO